A Parallel Implementation of Multiobjective Particle Swarm Optimization Algorithm Based on Decomposition

Jin-Zhou Li, *Wei-Neng Chen, *Member IEEE*, Jun Zhang, *Senior Member, IEEE*, Zhi-hui Zhan, *Member, IEEE*
Sun Yat-sen University, Guangzhou, China
Key Lab. Machine Intelligence and Advanced Computing, Ministry of Education, China
Collaborative Innovation Center of High Performance Computing, China
*Corresponding Author, Email: cwnraul634@aliyun.com

***Abstract*—Multiobjective particle swarm optimization based on decomposition (MOPSO/D) is an effective algorithm for multiobjective optimization problems (MOPs). This paper proposes a parallel version of MOPSO/D algorithm using both message passing interface (MPI) and OpenMP, which is abbreviated as MO-MOPSO/D. It adopts an island model and divides the whole population into several subspecies. Based on the hybrid of distributed and shared-memory programming models, the proposed algorithm can fully use the processing power of today's multicore processors and even a cluster. The experimental results show that MO-MOPSO/D can achieve speedups of 2x on a personal computer equipped with a dual-core four-thread CPU. In terms of the quality of solutions, it can perform similarly to the serial MOPSO/D but greatly outperform NSGA-II. An additional experiment is done on a cluster, and the results show the speedup is not obvious for small-scale MOPs and it is more suitable for solving highly complex problems.**

I. INTRODUCTION

A multiobjective optimization problem (MOP) involves several objective functions which require simultaneous optimization. For MOPs, the goal is to find a set of non-dominated solutions which is named as Pareto Set (PS) and the set of all the corresponding objective function values of all solutions in PS is called the Pareto Front (PF) [1]-[3]. During the last decade, multiobjective evolutionary algorithms (MOEAs) have become a popular research direction in evolutionary computation community, and have shown promising performance in MOPs [1]-[5].

Multiobjective Evolutionary Algorithm Based on Decomposition (MOEA/D), proposed by Qingfu Zhang [6], is an effective algorithm which adopts the main idea of that an approximation of the PF can be decomposed into a number of scalar objective optimization subproblems [7]-[11].

This work was supported in part by the NSFC projects Nos. 61379061, 61332002, 6141101191, in part by Natural Science Foundation of Guangdong for Distinguished Young Scholars No. 2015A030306024, in part by the "Guangdong Special Support Program" No. 2014TQ01X550, and in part by the Guangzhou Pearl River New Star of Science and Technology No. 151700098.

In real world applications, there exists some computational-intensive optimization problems that have a large number of decision variables (i.e., high dimension) and the computation of objective functions is time-consuming. Due to the nature of EAs that using a population to generate solutions iteratively, the computational cost of applying EAs to such computational-intensive problems become too high. Meanwhile, because of the inherent parallelism of EAs' population architecture and the rapid development of parallel and distributed computation, the parallel EAs attracted researchers' great attention.

Since most personal computers are now equipped with a multicore CPU, some parallel PSO algorithms are based on multicore CPUs with distributed memory model using MPI and several parallel PSO implementations are based on OpenMP or NVIDIA's GPUs with shared memory model. MPI is a cross-language communication protocol and it defines some message passing interface. OpenMP is an API that supports shared memory multithreading programming.

V. Roberge [12] implemented a basic MPI-PSO and K. Deep [13] presented three versions of parallel PSO based on MPI. K. Y. Tu [14] proposed a multithreading PSO computation model and Y. Hung [15] introduced a GPU-PSO algorithm. In the applications, parallel PSO algorithms are used to solve the biomechanical system identification problem [16] and the design optimization of composite structures [17]. For MOPs, Y. Zhou and S. Solomon proposed two parallel MOPSO algorithms using GPU respectively [18] – [19]. However, there is no research about MOEA/D's parallelism and there is no parallel PSO implementation using hybrid of MPI and OpenMP that fully uses the processing power of today's multicore processors with Hyper-Threading technology. For these reasons, we implement a parallel version of it using MPI and OpenMP (MO-MOPSO/D) based on the original MOPSO/D. Compared to the original algorithm, our proposed MO-MOPSO/D has the following features.

- The algorithm adopts an island model and the entire population is divided into several subspecies based on

978-1-4799-7560-0/15 $31.00 © 2015 IEEE
DOI 10.1109/SSCI.2015.187

the needed number of their neighbors in the algorithm. Each subspecies is regarded as an island and the all particles in an island are neighbors to each other.

- Both MPI and OpenMP are used to implement the algorithm. The evolution of one subspecies is computed by one process and the updating of a particle of the subspecies is done by one thread, which can fully use the CPU's power.
- In order to further reduce the MPI's communication cost, we only transfer the information of two particles at left and right edges of a subspecies to its left and right neighbors, respectively.

We run MOPSO/D and MO-MOPSO/D on a multicore PC and a cluster, separately. The results show MO-MOPSO/D can achieve a speedup of 2× while maintain the nearly same quality of the final solutions compared with MOPSO/D.

This paper is organized as follows. In Section II, a brief review on MOEA/D and its one implementation based on PSO (MOPSO/D) are presented. Section III introduces MO-MOPSO/D. Section IV compares MO-MOPSO/D with MOPSO/D. Section V presents more improvements on MO-MOPSO/D. Section VI includes the discussion, conclusions, and future research.

II. MOPSO/D REVIEW

MOEA/D decomposes a MOP into N scalar optimization subproblems and solves them simultaneously. And there are several ways to decompose the problem. In this section, we first present a superior decomposing approach named Tchebycheff Approach [6] briefly. In the following, we introduce MOPSO/D algorithm employing Tchebycheff decomposition approach.

In this approach, let $\omega^1, \dots, \omega^N$ is a set of uniform spread of N weight vectors of m objectives, which are used to decompose the MOP into N scalar optimization subproblems. $\boldsymbol{\omega}^j = (\omega_1^j, \dots, \omega_m^j)^T$ for each j = 1, … , N, where $\omega_i^j \geq 0$ for all i = 1, … , m and $\sum_{i=1}^{m} \omega_i^j = 1$. Then a scalar optimization subproblem is of the form

$$g(x|\omega^j, \mathbf{z}^*) = minimize\ max\left[\omega_i^j |f_i(\boldsymbol{x}) - z_i^*|\right] \qquad (1)$$

$$\text{subject to } \mathbf{x} \in X$$

where $z_i^* = \max\{f_i(\boldsymbol{x}) | \mathbf{x} \in X\}$ and $\mathbf{z}^* = (z_1^*, \dots, z_m^*)^T$. z_i^* is the best fitness value named reference point found by the population so far. Each $\mathbf{x}$ that satisfies $f_i(\boldsymbol{x}) = z_i^*$ is a best solution for the i-th objective function. The optimal solution $\mathbf{x}$ are written gBest_i in MOPSO/D and a gBest with size of m is used to store all global best solutions found so far.

The sequential MOPSO/D works as follows:

Algorithm 1: MOPSO/D

Input: 1) MOP with m objective functions;
2) The n-dimensional search space X;
3) The number of subproblems(the number of population) N;
4) The uniform spread of N weight vectors $\omega^j, j = 1, \dots, N$;
5) The number of neighbors of a weight vector T;
6) A stopping criterion.

Output: All solutions of the population to N subproblems.

Step 1) Initialization

1.1) Initialize the gBest with Φ;

1.2) Select T neighbors for each particle based on the Euclidean distances between their weight vectors and store them into $B(p) = \{p_1, \dots, p_T\}, p = 1, \dots, N$;

1.3) Randomly initialize $\mathbf{x}^1, \dots, \mathbf{x}^N$ in the population;

1.4) Compute the initial objective fitness values $F(\mathbf{x}^i) = [f_1(\mathbf{x}^i), \dots, f_m(\mathbf{x}^i)]$;

1.5) Generate the initial reference point vector $\mathbf{z}^* = (z_1^*, \dots, z_m^*)^T$ by comparing the initial objective fitness values.

Step 2) Update

For i = 1 to N

2.1) Randomly select two neighbors from $B(i)$, and generate a new solution $\boldsymbol{y}$ using a genetic algorithm based on them and $\mathbf{x}^i$;

2.2) Produce a new solution $\boldsymbol{y}'$ by applying a heuristic improvement on $\boldsymbol{y}$;

2.3) If the reference point $z_j^* < f_j(\boldsymbol{y}')$ for each j = 1, … , m, then update $z_j^* = f_j(\boldsymbol{y}')$ and $\text{gBest}_j = \boldsymbol{y}'$;

2.4) For each neighbor j (j = 1, … , T) of particle i, if $g(\boldsymbol{y}'|\boldsymbol{\omega}^j, \mathbf{z}^*) < g(\mathbf{x}^j | \boldsymbol{\omega}^j, \mathbf{z}^*)$, then update $\mathbf{x}^j = \boldsymbol{y}'$;

End For

Step 3) Check Stopping Criteria

Stop the algorithm and output all optimal solutions if the stopping criteria is satisfied. Otherwise, go to Step 2).

III. MPI-OPENMP-BASED MOPSO/D

In order to fully use the processing power of multicore processors with Hyper-Threading technology, we propose a variant of MOPSO/D based on MPI and OpenMP (MO-MOPSO/D) to accelerate the computation of it. In this section, we analyze how to modify the original MOPSO/D algorithm to be a parallel implementation.

A. Network Topology and Parallelism

A mix of master-slave paradigm and peer-to-peer model can apply to the proposed parallel algorithm. Population are divided into S subspecies and every subspecies contains T ($T = N/S$) internal particles to solve T scalar optimization subproblems and K external particles (EP) to store the exchanged particles from two neighbor subspecies. Each subspecies is regarded as an island and the computation of it can be done by a single process. These T particles in a subspecies are neighbors to each other. In order to simplify the problem, we assume that N is divisible by S.

In the master-slave paradigm, Process 0 is the master node and each other process is a slave node. All global information of population should be stored and updated by Process 0 and all other steps can be synchronously executed should be done by each process in parallel.

Since a process does evolution computation for a subspecies and a thread does it for a single particle in a subspecies, both Step 1.3 and 1.4 of Algorithm 1 can be parallel executed on thread level, while Step 1.5 can only be parallel executed on process level.

In Step 2, since nearly every particle need exchange

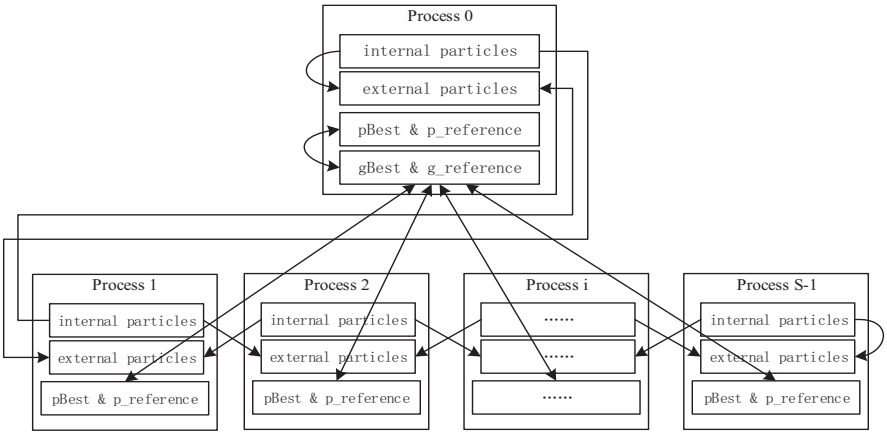

Fig. 1. Topology of MO-MOPSO/D. The lines with arrows represent the information transmission. A mix paradigm of master-slave paradigm and peer-to-peer model is applied to the architecture of MO-MOPSO/D.

Algorithm 2: MO-MOPSO/D

Input: 1) MOP with m objective functions;
2) The n-dimensional search space X;
3) The number of subproblems N;
4) The uniform spread of N weight vectors $\omega^j, j = 1, \ldots, N$;
5) The number of subspecies S;
6) The number of particles in a subspecies T, which is equal to N/S;
7) The number of particles to exchange between two neighbor subspecies K.
8) A stopping criterion.

Output: All solutions of the population to N subproblems.

Step 1) Initialization

1.1) Initialize the gBest of process 0 with Φ;
1.2) Initialize the sBest of all processes with Φ in parallel;
1.3) Process 0 calculates all weight vectors for population and distributes the needed components to each of the other processes using MPI_Scatter function.
1.4) All particles in a process are its neighbors for each particle and store them into $B(p) = \{p_1, \ldots, p_T\}$, $p = 1, \ldots, N$;
1.5) Randomly initialize $\mathbf{x}^1, \ldots, \mathbf{x}^S$ for each subspecies in parallel;
1.6) Compute the initial objective fitness values $F(\mathbf{x}^i) = [f_1(\mathbf{x}^i), \ldots, f_m(\mathbf{x}^i)]$ in parallel;
1.7) Update sBest and the subspecies reference point vector $\mathbf{s}^* = (s_1^*, \ldots, s_m^*)^T$ by comparing the initial objective fitness values in parallel.
1.8) Update gBest and the global reference point vector $\mathbf{z}^* = (z_1^*, \ldots, z_m^*)^T$ by comparing sBest.

Step 2) Update

For $i = 1$ to N
2.1) Swap particles with left and right neighbor subspecies for each subspecies and store them into EP.
2.2) Randomly select two neighbors from $B(i)$, and generate a new solution $\boldsymbol{y}$ by using evolution strategy in parallel;
2.3) Produce a new solution $\boldsymbol{y}'$ by applying a perturbation on $\boldsymbol{y}$;
2.4) If the subspecies reference point $s_j^* < f_j(\boldsymbol{y}')$, for each $j = 1, \ldots, m$, then update $s_j^* = f_j(\boldsymbol{y}')$ and $\text{sBest}_j = \boldsymbol{y}'$;
2.5) Update gBest and the global reference points.
2.6) For each neighbor j ($j = 1, \ldots, T$) of particle i, if $g(\boldsymbol{y}' | \boldsymbol{\omega}^j, \mathbf{z}^*) < g(\mathbf{x}^j | \boldsymbol{\omega}^j, \mathbf{z}^*)$, then update $\mathbf{x}^j = \boldsymbol{y}'$;
End For

Step 3) Check Stopping Criteria

Stop the algorithm and output all optimal solutions if the stopping criteria is satisfied. Otherwise, go to Step 2).

solution information with other particles in the different processes, the communication costs between particles are enormous. Thus, the topology of information transmission of particles can be revised to reduce communication costs on the basis of ensuring the optimization results.

In order to reduce communication costs between particles in neighbor processes, each process maintains K external particles (EP) copied from neighbors. Half of EP come from the left neighbor process and the others come from the right one. If a process has no left or right neighbor, this process should copy its first or last $K/2$ particles and store them into the corresponding positions in EP, respectively. In this sense, all processes are peers. The network topology of MO-MOPSO/D is represented in Figure 1.

The proposed parallel algorithm uses not only two random neighbor particles but EP to generate a new solution $\boldsymbol{y}$. Then a perturbation operator is employed to help the particle to escape from local best and obtain a newer solution $\boldsymbol{y}'$. Both of the two steps can be parallel executed for every particle on a thread level and the generation strategy is specifically introduced in Part C.

Like gBest, a subspecies best (sBest) should be maintained by each subspecies to lead the particles to fly. So the sBest and the corresponding subspecies reference point are updated after generating a new solution every time. The gBest and global reference point, computed by comparing all sBests, can be updated every several iterations to reduce communication costs. If a neighbor's current solution is worse than the new solution of the particle, then update it with the new solution. The update of sBest and gBest can be executed on process level while updating neighbors can be done in thread level, which is more efficient.

At last step, the algorithm checks if the stopping criteria is satisfied. If it is satisfied, then stop it and output results; otherwise, go to Step 2.

The algorithm is presented in Algorithm 2 and the flow chart of MO-MOPSO/D is shown in Figure 2.

B. Particles Exchange Strategy

As is shown in Figure 1, each subspecies exchanges its K particles with two neighbors. For one subspecies, its left half particles are close to its left neighbor subspecies while its right half particles are close to its right neighbor. Thus, when it exchanges particles with its neighbors, $K/2$ particles which are sent to its left neighbor should be randomly selected from its left half particles and the other $K/2$ particles that are sent to its right neighbor should be randomly selected from its right half particles.

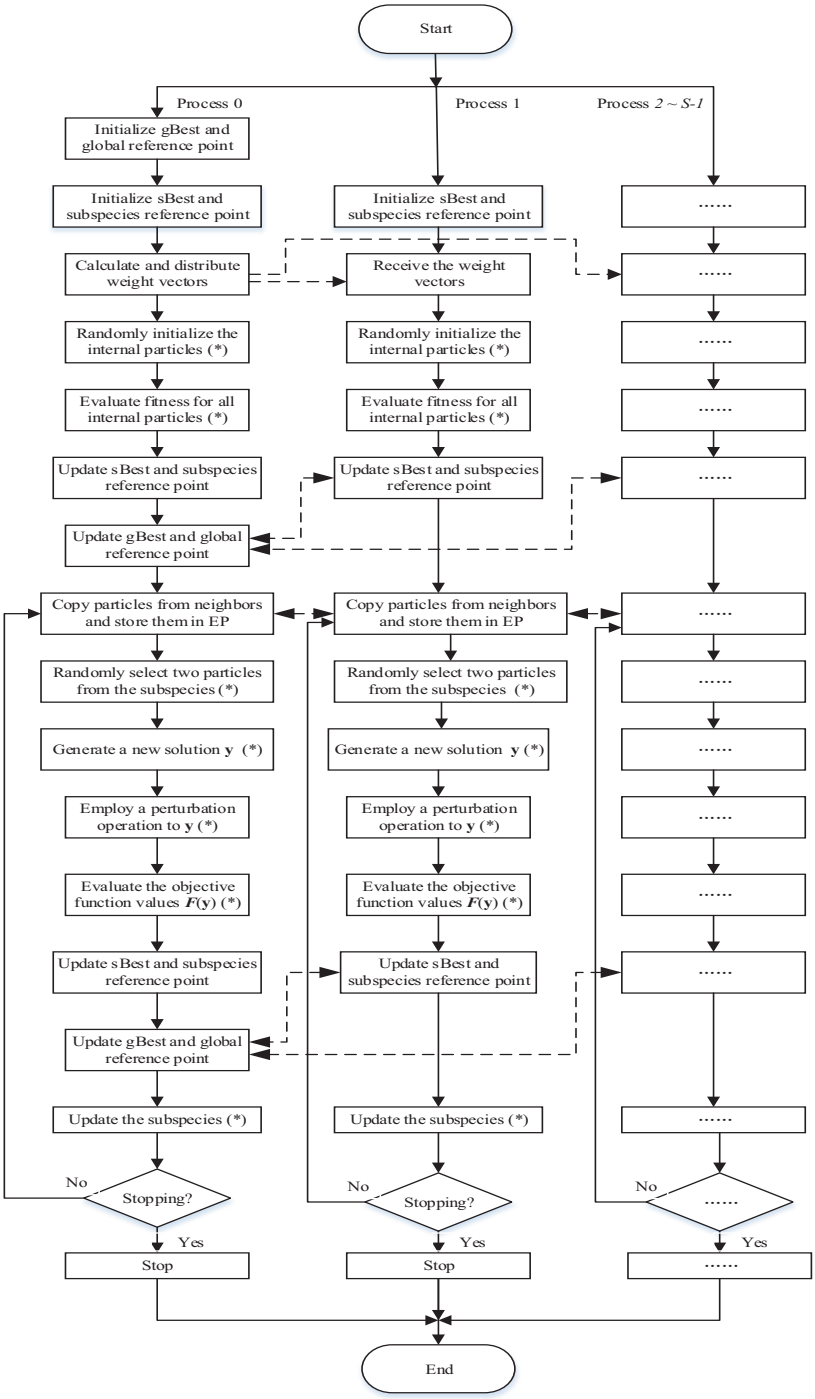

Fig. 2. Flow chart of MO-MOPSO/D. The solid lines represent the logical flow and the dashed arrows represent the information transmission. A step marked with a star means it can execute in parallel on thread-level.

After selecting the particles to exchange, there are two communication model to achieve the information transmission using MPI: one is point-to-point communication and the other one is collective communication. The point-to-point communication manner is intuitive that Process *i* send its information to the target processes, then the target processes receive the information. However, there are several disadvantages about point-to-point communication manner using MPI.

- An unreasonable order of sending and receiving information for processes can always cause system deadlock. But it is extremely difficult to design a reasonable flow of communication when there are many point-to-point communications between the processes.
- Compared with collective communications, point-to-point communications are inefficient, particularly when there exist extensive point-to-point communications.
- Programming using point-to-point communications is more complex, cumbersome and error-prone.

Based on above points, we adopt collective communications to achieve the particles exchange between neighbor subspecies in the proposed algorithm. In the master-slave communication paradigm employed in this paper, Process 0 is the master node and the other processes are slave nodes.

C. Selecting Parent Particles

In the island model of MO-MOPSO/D, all particles in an island are regarded as its neighbors for each other. However, the true nearness relationships among all particles in the population are defined by the distances between their weight vectors of their subproblems. And the optimal solution of two near subproblems should be similar. In order to avoid particles in an island losing subspecies diversity and falling into a local best point, the exchanged particles and two randomly selected neighbors are used to generate a new solution for every particle.

It is a more reasonable strategy in particle generation that selecting parent particles from neighbor particles or exchanged particles and even selecting which one from the whole exchanged particles should be based on the position of the particle in an island. At the first step, a probability Pr is defined based on its position to decide where to select the parent particles used to generate a new solution.

$$Pr = \begin{cases} \frac{T/2-p}{T-p} & if\ p < T/2 \\ 0 & if\ p = T/2 \\ 1-\frac{T}{2p} & if\ p > T/2 \end{cases} \quad (2)$$

where p = 0, … , T-1 is the position of the particle in its subspecies and T is the number of particles in a subspecies defined in part A. So, a parent particle is selected from EP and the other one is selected from two randomly selected neighbors if a random value is less than *Pr*, whereas the parent particles are both selected from neighbors at the rate of 1 - *Pr*.

At the second step, if one parent particle should be selected from EP, it is decided by the particle position that selecting the particle from the first or last half of EP. If $p < T/2$, the target region is the first half of EP, otherwise, that is the last half of it. Then randomly select a particle from the target region as a parent.

D. Discussions

1) How to set T and K: T is the number of particles in a subspecies and K is the number of exchanged particles between two neighbor subspecies. In the proposed algorithm, only the exchanged particles and neighbor particles are used to generate a new solution for a subproblem. So the setting of T and K is important for the optimal solutions. If T is too small, two neighbor particles selected in Step 2.2 may be too similar and the offspring particles will quickly converge to a local best point. Meanwhile, If T is too large, those two neighbor particles may be quite different from the current subproblem, so their offspring can hardly find the best solution of the subproblem. Since the exchanged particles coming from neighbor subspecies, it ensures the diversity of subspecies. However, if K is too large, much information need be transferred from one subspecies to another subspecies and the communication costs will be extensive. Actually, the EP with small K can lead to enough ability to explore the new solution areas.

2) Why the evolution of a subspecies is executed by a single process: In the MO-MOPSO/D, a process creates T threads and a thread executes the evolution of a particle. Based on the power of the processors with Hyper-Threading technology, two threads can run in a process simultaneously at least. However, the number of threads that a process allows to execute simultaneously is much less than the number of particles in a subspecies. So these threads are divided into several groups and they run in parallel within a group and run sequentially among the groups. In order to make all particle evolution can be executed in parallel, a subspecies must be divided into several parts and each part evolves on a separate process. The neighbors of a particle will lived in several islands and it will result in huge cost of communication when the particle needs the information of its neighbors living in a different island.

E. Comparision of Computational Complexity of MO-MOPSO/D and MOPSO/D

The major time consumption of the operators in MO-MOPSO is in Step 2. Step 2 generates N solutions to solve N scalar optimization subproblems in each generation. Note that Step 2 can be executed for each subspecies in parallel except Step 2.5. Step 2.1 just takes two particles from neighboring subspecies, Step 2.2 and Step 2.3 require $O(T)$ operations to generate trial solutions, Step 2.4 needs $O(mT)$ time to update sBest, Step 2.5 needs $O(mS)$ time to update gBest, and Step 2.6 need $O(mT^2)$ time to update **x**. Therefore, the ratio between the computational complexities of two algorithms is

$$\frac{O(mT^2)}{O(mNT)} = \frac{1}{S}$$

The MO-MOPSO/D has $1/S$ computational complexity of MOPSO/D.

IV. Comparision of MO-MOPSO/D With Mopso/D

In this section, five benchmark functions are applied to test the MO-MOPSO/D algorithm and MOPSO/D. Based on the results of the experiment, we evaluate the performance of MO-MOPSO/D and compare it with MOPSO/D.

A. Multiobjective Test Instances

Bi-objective ZDT test instances are widely used to test the performance of many MOEAs and the original MOPSO/D [20]. In order to compare their different performances between MO-MOPSO/D and MOPSO/D, five ZDT test instances are used in our paper. Since the experiment result can be significantly influenced by how to construct subspecies from population and this experiment focuses on the feasibility of parallelization of MOPSO/D.

B. Experimental Platform

To fairly compare the different performances of MOPSO/D and MO-MOPSO/D, we run these two algorithms on a personal computer, separately. The details of computer configuration are as follows.

- An Intel Core dual-core 4-thread i3-4150 CPU.
- 8 GB main memory.
- MS Windows 7 Professional operating system.

The above is the main hardware in this experimental personal computer. In terms of software, MPI and OpenMP are necessary for the experiment. MPICH2 v.1.4.1p1 for Windows are used to implement MO-MOPSO/D. The OpenMP version which shipped with VS2010 is used in the implementation of the proposed algorithm.

C. Performance Metrics

The speedup *Sp* is used to present the speed performance of a parallel algorithm and it is given by formula (3).

$$\mathrm{Sp} = \frac{T_{serial}}{T_{parallel}} \tag{3}$$

where T_{serial} and $T_{parallel}$ are the run-time of the serial algorithm and the parallel algorithm's run-time, respectively.

Inverted Generational Distance (IGD) [6] is applied in this paper to quantify the performances of solutions. IGD measures the distance between the real Pareto front and the approximate Pareto front obtained by an algorithm, and the distance is used to evaluate the convergence and diversity of the approximate PF. IGD can be calculated by formula (4),

$$IGD = \frac{1}{|P|}\sum_{i=1}^{|P|} Dist_i \tag{4}$$

where $Dist_i = min_{j=1}^{|A|}\sqrt{\sum_{m=1}^{M}\left(\frac{f_m(\mathbf{p}_i)-f_m(\mathbf{a}_j)}{f_m^{max}-f_m^{min}}\right)^2}$, f_m^{max} and f_m^{min} are the max and min values on the *m*-th object in set *P*, respectively. $m = 1, \ldots, M$, *P* is the real Pareto Solution Set and $\mathbf{p}_i, i = 1, \ldots, |P|$ is the *i*-th solution in *P*. *A* is the approximate Pareto Solution Set and $\mathbf{a}_j, \mathrm{j} = 1, \ldots, |A|$, is the *j*-th solution in *A*.

D. Experimental Setting

The size of population *N* is set to be 100 for both algorithms for all test instances. Because the information communication among subspecies is few in the island model, the speed of convergence of MO-MOPSO/D may decrease to a certain extent. In order to guarantee the quality of solutions, the number of particles generations needs to increase accordingly. Table I shows the number of generations for each test instance.

In order to reduce the communication costs between processes, the number of particles to exchange with two neighbor subspecies *K* is set to be 2. The number of particles in a subspecies *T* is set to be 20 and the weight vectors $\omega^j, j = 1, \ldots, N$ are set based on Section II and III. The size of the real Pareto Solution Set is set to be 1000 and the points in it are constructed uniformly.

The distribution indexes in crossover operator and the perturbation operator are both 20, the crossover rate is set to be 1.00 and mutation rate is set to be *1/D*, where *D* is the dimension of variables. For each test instance, the above two algorithms are executed 30 times independently.

E. Experimental Results

Table II shows the average wall clock time of MO-MOPSO/D and the average CPU time of MOPSO/D and the average speedup for each test instance. In general, MO-MOPSO/D can reduce the run-time for all test instances. MO-MOPSO/D runs about twice as fast as MOPSO/D for ZDT1, ZDT2 and ZDT3, and only about 1.2 times for ZDT4 and ZDT6. However, in consideration of the number of generations showed in Table I, MO-MOPSO/D can obtain a 3.2~3.5 times computing speedup for each instance, which is consistent with the dual-core 4-thread CPU used in the experiment.

TABLE I. Generations of Test Instances for two Algorithms

Test Instance	Number of generations in	
	sequential MOPSO/D	*MO-MOPSO/D*
ZDT1	300	500
ZDT2	300	500
ZDT3	300	500
ZDT4	500	1500
ZDT6	500	1500

TABLE II. Average Run-time and Speedup

Test Instance	Average run-time (in millisecond) used by		Speedup	Computing Speedup
	MOPSO/D	*MO-MOPSO/D*		
ZDT1	1404	715	1.96	3.27
ZDT2	1349	662	2.04	3.40
ZDT3	1391	703	1.98	3.30
ZDT4	1449	1266	1.14	3.43
ZDT6	1495	1294	1.16	3.47

TABLE III. COMPRARISION OF IGD AMONG THREE ALGORITHMS

Test Instance	MO-MOPSO/D	MOPSO/D	NSGA-II
ZDT1	1.32E-2	3.897E-3	1.892E-1
ZDT2	9.86E-3	3.826E-3	1.334E-1
ZDT3	7.76E-3	5.351E-3	1.003E-1
ZDT4	7.16E-3	4.051E-3	2.185E+0
ZDT6	8.10E-3	4.339E-3	1.426E-1

Table III shows the mean IGD obtained by MO-MOPSO/D, MOPSO/D and NSGA-II for each test instance. From Table III, the proposed algorithm gains a slightly better solution on ZDT3 and a little worse solutions on the other instances than MOPSO/D. Compared with NSGA-II, MO-MOPSO/D has great advantage in the quality of solutions.

From the above experiment results, we can conclude that the proposed algorithm achieves a good performance on both run-time and the quality of solutions compared with NSGA-II. However, in order to evolve in parallel, the island model is adopted in the algorithm and the entire population is divided into several subspecies, thus MO-MOPSO/D can only obtain slightly worse solutions than MOPSO/D while it only costs much less run-time than the latter.

V. A FEW IMPROVEMENTS ON MO-MOPSO/D

In Section IV, we compare the performance of MO-MOPSO/D and MOPSO/D, including average run-time and the quality of solutions obtained by these two algorithms. Revisiting Figure 3 and Table III, in terms of solution uniformness, the proposed parallel algorithm performs worse than the serial one on all test instances except ZDT3. So in this section, using a new particle selecting strategy, we improve the uniformness of solutions found by MO-MOPSO/D without efficiency degradation. On the other hand, we reduce the run-time of the parallel algorithm by running it on a cluster.

A. Deterministically Selecting a Particle to Exchange

As introduced in Section III-B&C, the left particle and the right one in a subspecies need be send to its left neighbor subspecies and right neighbor, respectively. Meanwhile, this particle can receive two particles from its left and right neighbors, which are used for the evolutions of particles. From this point of view, the exchanged particles have an enormous influence upon the final solutions found by the algorithm.

In the above experiment, we used a naive strategy that one particle randomly selected from the left half of a subspecies and the other one randomly selected from the right half of the subspecies are used as the exchanged particles. While here we deterministically select the first (left) particle and the last (right) particle from a subspecies, then send them to its left neighbor subspecies and right neighbor, respectively.

Table IV shows the mean IGD obtained by the improved MO-MOPSO/D for each test instance. Compared with the original MO-MOPSO/D, the improved MO-MOPSO/D reduces IGD and improves the quality of the final solutions. Fig. 3 represents the approximate Pareto front obtained by the improved MO-MOPSO/D for each test instance.

TABLE IV. COMPARISON OF IGD BETWEEN TWO PARALLEL ALGORITHMS WITH RANDOMLLY AND DETERMINISTICALLY SELECTING THE EXCHANGED PARTICLES

Instance	the Improved Parallel Algorithm	the Original Parallel Algorithm	MOPSO/D
ZDT1	1.04E-2	1.32E-02	3.897E-3
ZDT2	5.24E-3	9.86E-03	3.826E-3
ZDT3	5.02E-3	7.76E-03	5.351E-3
ZDT4	6.97E-3	7.16E-03	4.051E-3
ZDT6	5.25E-3	8.10E-03	4.339E-3

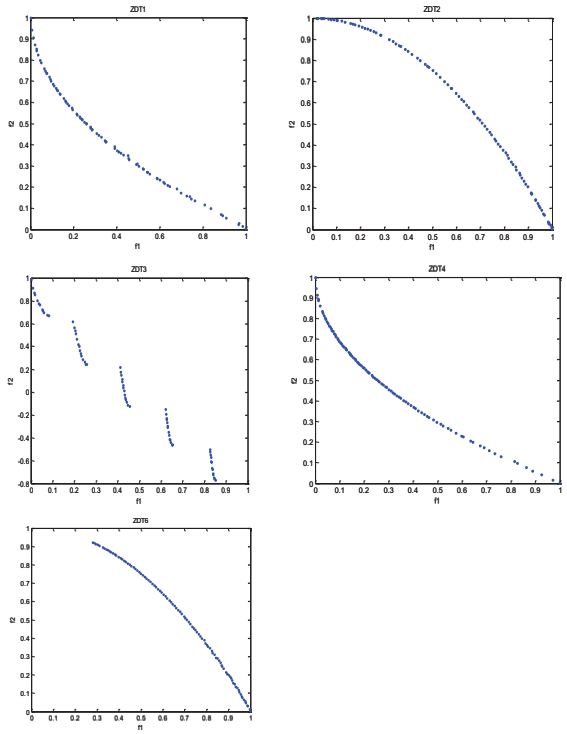

Fig. 4. The final solutions obtained by the improved MO-MOPSO/D for each test instance.

B. Running on the Cluster

To evaluate the performance of the parallel algorithm without being constrained by the number of processors in a PC, we execute the algorithm on a cluster. The specifications of the cluster are as follows.

- Five nodes, each is equipped with an Intel Core dual-core 4-thread i3-4150 CPU.
- 4 GB main memory per node.
- Gigabit network with FAST FS24 Switches
- Ubuntu Desktop 12.04 operating system.

Table V shows the average run-time of MO-MOPSO/D on the cluster (cluster MOPSO/D). Compared with the parallel MOPSO/D executed on a single machine, there is not significant decrease in the run-time for cluster MOPSO/D. Since the increase of communication costs on the cluster offsets the decrease of operation costs, so the total run-time is not greatly reduced. However, cluster can be highly necessary to solve the extremely complex multiobjective optimization problems.

TABLE V. AVERAGE RUN-TIME OF CLUSTER MOPSO/D AND SPEEDUP

Instance	Average run-time (in millisecond) used by			Speedup
	cluster MOPSO/D	*stand-alone MOPSO/D*	*serial MOPSO/D*	
ZDT1	626	715	1404	2.24
ZDT2	656	662	1349	2.06
ZDT3	677	703	1391	2.05
ZDT4	1075	1266	1449	1.35
ZDT6	1038	1294	1495	1.44

VI. CONCLUSION

Multiobjective Particle Swarm Optimization Based on Decomposition (MOPSO/D), an implementation of MOEA/D using PSO, is an effective algorithm for solving MOPs. In this paper, based on MPI and OpenMP parallel programming platforms, we proposed a parallel algorithm called MO- MOPSO/D. The algorithm combined distributed-memory and shared-memory programming models into an algorithm, which can fully use the processing power of multicore processors with Hyper-Threading technology and a cluster.

The proposed algorithm first decomposes the MOP into many scalar optimization subproblems, each of which can be solved by a single particle. Then the particles are decomposed into several subspecies based on the distances of their weight vectors. The evolution of a subspecies is executed by a single process and they communicate with each other by using MPI communication functions. In the process, threads are used to improve the evolution efficiency. In terms of the quality of solutions, the experiment results show MO-MOPSO/D can obtain the final solutions which are slightly worse than those of MOPSO/D but better than those of NSGA-II. As for the run-time, the algorithm can achieve speedups of 2x on the single PC equipped with a dual-core four-thread CPU.

Since the uniformness of solutions is not good, the deterministic particle selecting strategy is used in the improved parallel algorithm. The results have shown the strategy highly improved the uniformness of solutions. We have also executed the improved parallel algorithm on a cluster to reduce its run-time. And the result showed that running the algorithm on a cluster can boost its efficiency a bit for these simple test instances, since the communication costs is higher than its operation costs. However, it also suggested that a cluster is more suitable for the complex MOPs. Our future attempts will be in solving the complex MOPs on clusters based MO-MOPSO/D.

REFRENCES

[1] K. Deb, Multi-Objective Optimization using Evolutionary Algorithms. New York: Wiley, 2001.

[2] C. A. C. Coello, D. A. V. Veldhuizen, and G. B. Lamont, Evolutionary Algorithms for Solving Multi-Objective Problems. Norwell, MA: Kluwer, 2002.

[3] K. Tan, E. Khor, and T. Lee, Multiobjective Evolutionary Algorithms and Applications, ser. Advanced Information and Knowledge Processing. Berlin, Germany: Springer-Verlag, 2005.

[4] K. Deb, Multiobjective Optimization Using Evolutionary Algorithms, New York: Wiley, 2001.

[5] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans. Evol. Comput, vol. 6, no. 2, pp. 182-197, Apr. 2002.

[6] Q. Zhang and H. Li. MOEA/D: A Multiobjective Evolutionary Algorithm Based on Decomposition. IEEE Trans. on EvoL Comp., vol. 11 , pp. 712-731, 2007.

[7] L. Paquete and T. Stützle, A two-phase local search for the biobjective traveling salesman problem. Proc. Evol. Multi-Criterion Optim., 2003, pp. 479–493.

[8] E. J. Hughes, Multiple single objective Pareto sampling. Proc. Congr. Evol. Comput., Canberra, Australia, 2003, pp. 2678–2684.

[9] Y. Jin, T. Okabe, and B. Sendhoff, Adapting weighted aggregation for multiobjective evolutionary strategies. Evolutionary Multicriterion Optimization. Springer, 2001, vol. 1993, LNCS, pp. 96–110.

[10] H. Ishibuchi and T. Murata, Multi-objective genetic local search algorithm and its application to flowshop scheduling. IEEE Trans. Syst., Man, Cybern., vol. 28, pp. 392–403, Aug. 1998.

[11] A. Jaszkiewicz, On the performance of multiple-objective genetic local search on the 0/1 knapsack problem – A comparative experiment. IEEE Trans. Evol. Comput., vol. 6, no. 4, pp. 402–412, Aug. 2002.

[12] V. Roberge, M. Tarbouchi, Comparison of parallel particle swarm optimizers for graphical processing units and multicore processors. International Journal of Computational Intelligence and Applications, vol. 12, no. 01, 2013.

[13] K. Deep, S. Sharma, M. Pant, Modifid Parallel Particle Swarm Optimization for Global Optimization Using Message Passing Interface. Bio-Inspired Computing: Theories and Applications (BIC-TA), IEEE Fifth International Conference, pp. 1451-1458, 2010.

[14] K. Y. Tu and Z. C. Liang, Parallel computation models of particle swarm optimization implemented by multiple threads. J. Expert Syst. Appl. Vol. 38, no. 5, 2011, pp. 5858-5866.

[15] Y. Hung and W. Wang, Accelerating parallel particle swarm optimization via GPU. Optimization Methods and Software, 2012, pp. 33-51.

[16] J. F. Schutte, B. J. Fregly, R. T. Haftka and A. D. George, A Parallel Particle Swarm Optimizer. FLORIDA UNIV GAINESVILLE MECHANICAL AND AEROSPACE ENGINEERING, 2003.

[17] S. N. Omkar, A. Venkatesh, M. Mudigere, MPI-based parallel synchronous vector evaluated particle swarm optimization for multi-objective design optimization of composite structures. Engineering Applications of Artificial Intelligence, vol. 25, pp. 1611–1627, 2012.

[18] Y. Zhou and Y. Tan, GPU-Based Parallel Multi-objective Particle Swarm Optimization. International Journal of Artificial Intelligence, 2011, pp. 125-141.

[19] Proceedings of the 13th annual conference on Genetic and evolutionary computation. ACM, 2011.

[20] E. Zitzler, K. Deb, and L. Thiele, "Comparison of multiobjective evolutionary algorithms: Empirical results," Evol. Comput, vol. 8, no. 2, pp. 173–195, 2000.